Freeform Search

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Term:

Display: 10 **Documents in Display Format:** - **Starting with Number** 1

Generate: ○ Hit List ⊙ Hit Count ○ Side by Side ○ Image

Search Clear Interrupt

Search History

DATE: Wednesday, November 05, 2003 Printable Copy Create Case

Set Name side by side	Query	Hit Count	Set Name result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L31	db2 near tablespace	6	L31
L30	707.clas.	17392	L30
L29	707/204	1507	L29
L28	707/203	1980	L28
L27	707/202	1483	L27
L26	707/201	1911	L26
L25	707/200	2820	L25
L24	707/7	1323	L24
L23	707/102	3775	L23
L22	707/101	2746	L22
L21	707/100	3801	L21
L20	707/1	5455	L20
L19	L18 and log	21	L19
L18	L17 and backup near (copy or copies)	22	L18
	L16 and (database or data with base) near (recovery or recover or		

L17	recover$)	77	L17
L16	L15 and (tablespace or table or space or table-space)	1206	L16
L15	707/204	1507	L15
L14	6178427.uref.	5	L14
L13	6178427.pn.	2	L13
L12	L11 and (table-space or tablespace)	0	L12
L11	6044444.pn.	2	L11
L10	60444444.pn.	0	L10
DB=USPT; PLUR=YES; OP=OR			
L9	5903898.pn.	1	L9
L8	6044444.pn.	1	L8
DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR			
L7	"parker, christopher".in.	28	L7
L6	L5 and log	3	L6
L5	L3 and (backup or back adj up) near (copy or copies or replicate)	4	L5
L4	L3 and (backup or back adj up) near (copy or copies or replica)	4	L4
L3	L2 and (table with space or table adj space or table-space or tablespace)	33	L3
L2	L1 and table near (recover or recovery or recover$)	120	L2
L1	(database or data adj base or data with base)	410101	L1

END OF SEARCH HISTORY

First Hit Fwd Refs

Generate Collection Print

L31: Entry 4 of 6 File: USPT May 14, 1996

DOCUMENT-IDENTIFIER: US 5517641 A
TITLE: Restartable method to reorganize DB2 tablespace records by determining new physical positions for the records prior to moving using a non sorting technic

Abstract Text (1):
An improved method to dramatically reduce the time required to reorganize DB2 tablespaces and index files by not utilizing conventional sort techniques. Viewing access is allowed during the reorganization process by setting the files to read only status. The process is basically non-destructive, allowing a prompt return to the original state, and is checkpointed, allowing restarting at selected intervals. Briefly, the original table and indices are considered as A files and read into memory. New row IDs or RIDs are developed using a non-sorting technique so that the proper order of the data is developed. After the new RIDs have been developed, both the clustering index and the row data are read out of memory and written to a new table and clustering index files in the proper order as B files. Then any remaining non-clustering indices are reorganized using non-sorting techniques in a similar fashion.

Brief Summary Text (3):
The invention relates to reorganizing database data and index files, particularly DB2 tablespaces, into key order without utilizing conventional sorting procedures, while allowing the tablespaces to be viewed during reorganization and allowing prompt recovery or restarting of the process if interrupted before completion.

Brief Summary Text (6):
A tablespace is the DB2 term used to identify a database. Tablespaces can be simple, segmented or partitioned. In a simple tablespace, the data is kept in one file and there may be a clustering index and other indices. A clustering index is the index where the keys are kept in sequential order and the data is preferably kept in this same order. In a segmented tablespace, many different logical data files or tables are kept in a single file and there may be a clustering index and other indices for each logical data file. There are no indices directed to multiple logical files. In a partitioned tablespace, the data is kept in different files, but there is a clustering index for each file. There may be additional indices directed to all of the partitions.

Drawing Description Text (3):
FIG. 1A and 1B are a flowchart of a DB2 tablespace reorganization procedure according to the present invention;

Detailed Description Text (54):
Further, the above description has focused on DB2 tablespaces, but similar techniques can be used for databases developed in other formats, particularly tree-structured formats.

First Hit
End of Result Set

☐ Generate Collection Print

L31: Entry 6 of 6 File: TDBD Mar 1, 1994

TDB-ACC-NO: NN940391

DISCLOSURE TITLE: Single-Source Reference/Multiple Table Access via DB2 Synonyms

PUBLICATION-DATA:
IBM Technical Disclosure Bulletin, March 1994, US

VOLUME NUMBER: 37
ISSUE NUMBER: 3
PAGE NUMBER: 91 - 92

PUBLICATION-DATE: March 1, 1994 (19940301)

CROSS REFERENCE: 0018-8689-37-3-91

DISCLOSURE TEXT:

The ImagePlus* Batch processes need to be able to access and update data which reside in, what has been termed by the ImagePlus MVS/ESA* platform, TABLESETs. This construct means that data from various applications, which comprise the same DB2 table attributes, are stored and managed in "mirror" tables. This was done for many reasons, but mainly to allow for: o a separation of data by application, and o the need for customers to store greater than 64 gigabytes of data (e.g., capacity limitation on DB2 tablespaces). - The batch processes run in a CAF environment which will allow a single task to connect to a DB2 subsystem, open a DB2 plan (a thread connection to DB2) that should be opened, and process data related to the plan. A single plan contains the access paths and pointers to a "SET" of "TABLES," thus the term TABLESET. But another plan contains a complete set of the same tables WITH DIFFERENT NAMES. - In our batch process design and implementation, we accommodate all tablesets via ONE set of code pointing to a single reference. For example, our code might appear as: o SELECT * FROM TABLE00 Based upon DB2 synonyms, which are USERID oriented, we need to BIND a plan that contains the DBRM for the above SELECT statement, and point the reference "TABLE00" at a REAL table within a tablespace. This plan gets utilized when it is OPENed within the CAF routine via DSNALI (DB2 language interface). In order to accomplish the desired result of this plan pointing to the correct REAL tablespace, we employed the following scenario: 1. SET CURRENT SQLID to user1 2. CREATE SYNONYM TABLE00 For USER1.realtable 3. Bind Plan with owner of User1 Thus, whenever this plan is active, TABLE00 will be dealing with User1.realtable. You can see whereby establishing any number of these could allow the same code to deal with many different tables. Since CAF within the application controls the plan to be opened, our design accommodates the use of many tables with one set of codes, simply by creatively using the synonym and bind processes to accomplish the end result. * Trademark of IBM Corp.

SECURITY: Use, copying and distribution of this data is subject to the restictions in the Agreement For IBM TDB Database and Related Computer Databases. Unpublished - all rights reserved under the Copyright Laws of the United States. Contains confidential commercial information of IBM exempt from FOIA disclosure per 5 U.S.C. 552(b)(4) and protected under the Trade Secrets Act, 18 U.S.C. 1905.

COPYRIGHT STATEMENT: The text of this article is Copyrighted (c) IBM Corporation 1994. All rights reserved.

Refine Search

Search Results -

Terms	Documents
(database or data with base) and (table with space near recovery or tablespace near recovery)	4

Database:
US Pre-Grant Publication Full-Text Database
US Patents Full-Text Database
US OCR Full-Text Database
EPO Abstracts Database
JPO Abstracts Database
Derwent World Patents Index
IBM Technical Disclosure Bulletins

Search:

Refine Search

Recall Text Clear Interrupt

Search History

DATE: Monday, November 03, 2003 Printable Copy Create Case

Set Name side by side	**Query**	**Hit Count**	**Set Name** result set
	DB=PGPB,USPT,USOC,EPAB,JPAB,DWPI,TDBD; PLUR=YES; OP=OR		
L5	(database or data with base) and (table with space near recovery or tablespace near recovery)	4	L5
L4	L3 and (recover or recovery)	17	L4
L3	(database or data with base)near (table with space or tablespace)	41	L3
L2	mirror$ near log near2 (data with sets or datasets)	1	L2
	DB=USPT; PLUR=YES; OP=OR		
L1	6189010.pn.	1	L1

END OF SEARCH HISTORY

L8: Entry 4 of 9 File: USPT Dec 12, 2000

DOCUMENT-IDENTIFIER: US 6161109 A
TITLE: Accumulating changes in a database management system by copying the data object to the image copy if the data object identifier of the data object is greater than the image identifier of the image copy

Brief Summary Text (5):
Databases are computerized information storage and retrieval systems. A Relational Database Management System (RDBMS) is a database management system (DBMS) which uses relational techniques for storing and retrieving data. Relational databases are organized into tables which consist of rows and columns of data. The rows are formally called tuples. A database will typically have many tables and each table will typically have multiple tuples and multiple columns. The tables are typically stored on direct access storage devices (DASD) such as magnetic or optical disk drives for semi-permanent storage.

Brief Summary Text (6):
A table is assigned to a tablespace. The tablespace contains one or more datasets. In this way, the data from a table is assigned to physical storage on DASD. Each tablespace is physically divided into equal units called pages. The size of the tablespace's pages is based on the page size of the bufferpool specified in the tablespace's creation statement. The bufferpool is an area of virtual storage that is used to store data temporarily. A tablespace can be partitioned, in which case a table may be divided among the tablespace's partitions, with each partition stored as a separate dataset. Partitions are typically used for very large tables.

Brief Summary Text (7):
A table may have an index. An index is an ordered set of pointers to the data in the table. There is one physical order to the rows in a table that is determined by the RDBMS software, and not by a user. Therefore, it may be difficult to locate a particular row in a table by scanning the table. A user creates an index on a table, and the index is based on one or more columns of the table. A partitioned table must have at least one index. The index is called the partition index and is used to define the scope of each partition and thereby assign rows of the table to their respective partitions. The partition indexes are created in addition to, rather than in place of, a table index. An index may be created as UNIQUE so that two rows can not be inserted into a table if doing so would result in two of the same index values. Also, an index may be created as a CLUSTERING index, in which case the index physically stores the rows in order according to the values in the columns specified as the clustering index (i.e., ascending or descending, as specified by the user).

Brief Summary Text (8):
RDBMS software using a Structured Query Language (SQL) interface is well known in the art. The SQL interface has evolved into a standard language for RDBMS software and has been adopted as such by both the American National Standards Institute (ANSI) and the International Standards Organization (ISO). The SQL interface allows users to formulate relational operations on the tables interactively, in batch files, or embedded in host languages, such as C and COBOL. SQL allows the user to manipulate the data. As the data is being modified, all operations on the data are logged in a log file.

Brief Summary Text (9):
One technique for recovering a database involves restoring a prior full image copy of the data and then reapplying subsequent logged changes to make the data current in time. Typically, the database containing partitions and indexes is stored on a data storage device, called a primary data storage device. The partitions are periodically copied to another data storage device, called a secondary data storage device, for recovery purposes. In particular, the partitions stored on the primary data storage device may be corrupted, for example, due to a system failure during a flood, or a user may want to remove modifications to the data (i.e., back out the changes). In either case, for recovery, the partitions are typically copied from the secondary data storage device to the primary data storage device. Next, using the log file, the copied data is modified based on the operations in the log file. Then, the indexes are rebuilt. In particular, to rebuild the indexes, keys are copied from each row of each partition, sorted, and then used to create a partition index. Additionally, the table index is rebuilt based on the partition indexes.

Brief Summary Text (10):
Another technique for recovering a database involves restoring the database using a prior full image copy, restoring one or more partial image copies (sometimes called incremental copies), and then reapplying subsequent logged changes to make the data current in time. The partial copies contain accumulated changes made to the data since the previous full or partial image copy operation. In some systems, the changed data is identified using indicators (i.e., usually called "dirty" bits or "status" bits) associated with each record or block of records (i.e., sometimes called a "page" of records) to designate that a change has occurred to a record or block. Whenever the record or block is first modified, the indicator is set. Each time the record or block is placed in an image copy, the indicator is reset. However, the overhead of maintaining these indicators is significant. With high transaction loads in a data sharing complex, providing coherency of these indicators across the records and blocks of the complex can result in degraded transaction and system performance.

Detailed Description Text (11):
The image copying system 122 in conjunction with the data manager 118 provides a technique for determining which data from a database has been modified and should be copied from an "original" or "primary" copy on a primary data storage device to update an "image" or "secondary" copy on a secondary data storage for use as a backup copy device. The image copying system 122 is able to make the determination without the use of "status" bits. The image copying system 122 delays the need to determine whether a record or block has been updated until the time at which image copies are being made. Additionally, the image copying system 122 preferably makes the determination using log sequence numbers (LSNs).

Detailed Description Text (12):
FIG. 2 illustrates a sample database and its image copy that could be used in accordance with the present invention. The database 200 is stored on a primary data storage device. The database 200 contains "data objects" 202 (e.g., records or blocks). Application programs 204 modify the database 200 by adding, updating, or deleting data via operations. These modifications are logged in a log file. Periodically, the database 200 or a portion of the database 200 is copied to a secondary data storage device 206. A full image copy involves copying all of the data in the database 200 to the secondary data storage device 206. A partial image copy involves copying data that has been modified since the last copy operation to the secondary data storage device 206. The image copying system 122 makes a determination at the time of making a copy of which data has been modified since the last copy was made. The image copying system 122 makes this determination by using log sequence numbers or comparable means, for example, a timestamp or other indicator of the sequence of the modifications in time relative to backup copies and partial copies. Each log sequence number relates to a point in time and is used

to indicate which of two data objects was modified most recently. The log sequence number is typically a monotonically increasing number.

Detailed Description Text (17):
Additionally, an image copy backup LSN ("ICBU LSN") is assigned to the image copy. The image copying system 122 records an initial ICBU LSN in the image copy and in a system catalog. The ICBU LSN is also based on the global LSN. At the time a record or block is copied, the ICBU LSN for the entire copy (i.e., not just for a particular record or block) is updated to match the most recent global LSN. When a global LSN is used in this manner, the global LSN is incremented.

First Hit Fwd Refs

End of Result Set

Generate Collection Print

L8: Entry 9 of 9 File: USPT Jul 31, 1990

DOCUMENT-IDENTIFIER: US 4945474 A
TITLE: Method for restoring a database after I/O error employing write-ahead logging protocols

Brief Summary Text (2):
In the development of information processing systems, relational database management programs evolved allowing the user to search, access and alter data contained in numerous different database tables by using specific fields common to all such tables.

Brief Summary Text (5):
Perhaps one of the most obvious general approaches to the problem was to provide for redundancy whereby backup copies of the data were available in the event that the database or portions thereof needed to be reconstructed due to such incomplete log writes or detected log write failures. Accordingly, several techniques were developed in the art for providing such redundancy, one of the earliest being known as shadow paging which essentially involves retaining a copy of an entire page of data while updates were made to a second copy. After the newer copy containing the changes was safely written to the permanent medium, the archival copy could thence be written over. This technique was employed for example in the database product of the IBM Corporation known commercially as SQL/DS. A survey of various systems employing this shadow copy technique may be found in "File Servers for Network Based Distribution Systems", Liba Svobodova, ACM Computing Surveys, Vol. 16, No. 4 (December 1984), pages 353-399.

Brief Summary Text (6):
Although shadow paging appeared to be a viable solution in some environments it was not without its disadvantages including the expense and space involved in maintaining such shadow copies. Accordingly, database systems began implementing the transaction recovery facility by only writing changes to database records to both the changed record and to a database recovery log. The information recorded in the database recovery log insured that changes of committed transactions were incorporated into the database state during system restart following a system failure (as well as allowing changes to database records to be reversed or undone in support of transaction rollback for uncommitted transactions).

Brief Summary Text (8):
Information contained in index nodes of these index files was extremely important in providing key record information that was frequently deleted or inserted as records were deleted or inserted into the database tables, and consequently such concurrent accessibility on a sub-page level was highly desirable. A particularly important aspect of such index files was that individual fields of a record in a database might frequently logically contain data which was not kept in the record for itself but rather, (by means of a pointer or descriptor) kept in a separate file. Example of such a file is known as a long field file, wherein a long field is contained, which may have an image associated with large data set type items such as audio or image data which can be extremely valuable, thus illustrating the importance of such indexes.

Brief Summary Text (9):
With the foregoing in mind, it will be appreciated that it was desirable to provide for a database recovery system of the write-ahead logging type which nevertheless provided for such sub-page level concurrency. Systems were accordingly developed such as those described in U.S. Pat. Application Ser. No. 07/059,666, filed Jun. 8, 1987, and entitled "Method for Managing Sub-Page Concurrency Control and Partial Transaction Rollback in a Transaction-Oriented System of the Write-Ahead Logging Type", now abandoned, and refiled On Sept. 7, 1989, as pending continuation Ser. No. 07/406,186, as well as pending U.S. Pat. Application Ser. No. 07/115,146, filed Oct. 30, 1987, and entitled "Method for Concurrent Record Access Using an Index Tree", U.S. Pat. No. 4,914,569. An additional reference that discusses these index files such as those commonly configured in a B-tree structure known in the art is "Efficient Locking for Concurrent Operation on a B-Tree"by Lehman and Yao, ACM Transactions on Database Systems, Vol. 6, No. 4, (December 1981), pages 650-670, the hereinbefore noted references being incorporated herein by reference.

Brief Summary Text (10):
Notwithstanding the aforementioned advances, problems nevertheless remained in providing for effective database recovery First on restart processing of such systems, files with I/O errors were not readily detectable so as to prevent and safeguard restart operations from accessing the files with attendant data loss. Further, means were not provided for readily detecting incomplete log writes or detected log write failures in order to stop the further writing of transactions. Moreover, no effective means was provided for readily identifying such error files during restart. Additionally, rebuilding of error file indexes was by no means automatic but rather required explicit user action and invalidated access plans related to the failing index.

Brief Summary Text (11):
Accordingly, systems and methods were desired for reducing data loss due to I/O errors and power failure during non-atomic writes to disk in a transaction management system using write-ahead logging protocol. Such systems and methods were highly sought whereby I/O error on index files, including system tables, caused no data loss. Also, techniques were desired for providing automatic recovery from the errors without an explicit user action to rebuild the affected indexes. Means were desired whereby power failure during log file writes caused no data loss without the necessity for employing double writes, shadow paging or the like. It was further highly desired to provide effective means whereby I/O error on user tables had limited data loss effect to the table in error. Additionally, it was desirable to provide a technique for index file rebuilds which did not invalidate the access plans related to the index. These and other desired features not met by the prior art are provided by the subject invention as hereinafter described in greater detail.

Drawing Description Text (3):
FIG. 1 is an illustration of a database table;

Drawing Description Text (4):
FIG. 2 is an illustration of data pages of the database table of FIG. 1 conceptually as they would be stored in media;

Drawing Description Text (5):
FIG. 3 is an illustration of an index B-tree for the database table of FIG. 1 as it would be stored conceptually in media;

Drawing Description Text (6):
FIG. 4. is an illustration of long field file data corresponding to the employees in the table of FIG. 1;

Detailed Description Text (2):
In order to better understand the invention, first with reference to FIGS. 1-4 a more detailed description of representative data and the manner in which it is stored in a database system will be given using the illustration of the employee table of FIG. 1. With reference to FIG. 3, an example will be given of the concept of a database index. Next, with reference to FIG. 4, the use of such index information of FIG. 3 will be illustrated in accessing a particularly important form of files shown in FIG. 4, i.e., long field files. As will become more apparent, these indexes actually reference corresponding data records which in turn reference correlative long fields.

Detailed Description Text (4):
FIG. 1 conceptually represents an example of actual data that a user may have entered in a database. This data is essentially in this example a list having headers such as employee name 10, some form of employee number 12, and some type of image data 14 such as a corresponding employee picture. Thus, the first record in this table is Andrew, employee number 1, and data comprising a picture of Andrew. Similar records appear for other employees entered into the database.

Detailed Description Text (5):
FIG. 2 is an illustration of the file representing part of the table data of FIG. 1 as it might be stored on computer disk. The file represents, sequentially, data pages such as pages 0-2 (reference numerals 16, 18, and 20, respectively) each having some of the records of the FIG. 1 table. Each data page has a header and trailer. Page 0, for example, has 22 for the header and 30 for the trailer; page 1 has 36 and 44; and 20 has 50 and 58 for the header and trailer, respectively. The header and trailer each contain a log sequence number or LSN. This LSN is copied from the header to the trailer prior to writing out a page. After reading back a page, the header and trailer are compared to make sure that they are identical in order to verify that the previous write was completed.

Detailed Description Text (9):
In relational databases, it is necessary to efficiently access the data such as that of Table 1, FIG. 1. This is conventionally done by means of an index file, an example of which is schematically illustrated (for the Table 1 data) in FIG. 3, in a manner well known in the art. FIG. 3 depicts an index for Table 1 wherein the index is in the form of a two level tree. The root level on page 3 (64) of the database file has pointers to the three leaves or nodes 66, 68 and 70. These nodes have index data stored in pages 5, 4, and 6 of the index file, respectively. D, F, and "null" in page 3 (64) of the index file each represent the highest possible "key" in the particular node 66, 68, 70 to which they point, respectively. In this case, a key is the first alphabetic character of the employee name although in other applications wherein numeric data is stored the keys may be numbers. Andrew, Baker, and Chester have first alpha characters less than D and thus indexing data related to them are stored in node 66. F in root 64 points to node 68 which contains Edgar and Edward's indexing data because their first alpha characters are less than F but greater than D. The null pointer of root 64 is the highest allowable key and thus points to the last leaf 70. Leaf 70 contains Howell since its first alpha, H, is greater than F (the highest key in the preceding node 68).

Detailed Description Text (24):
FIG. 7 is a flow chart illustrating the processing of log pages and the handling of an error detected during database restart processing. 134 is the start of this processing wherein at 136 the nth log page is being read in. If, after reading in the log page, it is determined at 138 that there is an I/O error or a mismatch of the header and trailer LSNs, then at 172 the next log page n+1 is read in. If there is I/O error or mismatched LSN, determined at 174, then an error status is returned, 180. In this particular case at 180 the database could not be recovered and must thence be restored from a backup copy. If, at 174, there is no I/O error or mismatched header-trailer LSN's, then decision block 176 is reached. If the LSN

of page n+1 is less than the start LSN of the log, then it is known that this page has not previously been written out and the process proceeds to 178. At 178 end of log file status is returned, indicating processing of all pages has been completed.

Detailed Description Text (28):
FIG. 8 is a flow chart illustrating the detection and processing of an error on any data file, index file, or long field file during database restart processing. Processing starts at 182 wherein beginning of processing a log record occurs. The log record, it will be recalled, identifies the page and file from which to read the particular desired data, index or long field file data. At 184, a page specified from the log record is read. At 186, the check is made for any read error. Upon read error detection, at 190 the check is made to see if the file wherein the read is being attempted exists. If not, at 196 a successful operation is returned. If the file does exist, then at 194 the file is renamed and again a successful operation is returned at 196. Returning to 186, if no read error occurs, a check is made at 188 to determine whether a mismatched header/trailer LSN exist. If so, again at 194 the file is renamed and a successful operation is returned at 196. If, on the other hand, at 188 no mismatched header/trailer LSN is detected, operations will then be performed at 192 which would have normally occurred, these operations being whatever the log record requires to be done at this point. This may, for example, be the redoing of a particular operation, adding a key if required at the time, or the like. Finally, with reference to FIG. 8, after recovery, error file identifying information can be captured and placed in a table or on disk. A preferred method, however, would be to store such information in a bit map.

Detailed Description Text (31):
Returning now to FIG. 10, this is a flow chart illustrating error index file processing during normal non-database restart. At 218 a check is made of whether a request has been made to open a table. If not, then an error message is returned at 220. An open table request is a request to begin processing of a new table. If such a request is received by the process, then at 222 the procedure is begun to recreate the index file. The beginning of such a procedure at 222 is to delete the error index file. Once this has been done at 224 the first index definition is located using the system index table. Using this first definition at 226 the process begins to create or recreate the index as defined in 224. At 228 a check is made of whether all indexes have been created yet. If not, at 230 the next index definition is retrieved and a loop back to 226 occurs wherein that fetched index is recreated. Again continuing to check 228, if all indexes have been created, then at 232 the "open table" request is completed and a successful return at 234 occurs.

Other Reference Publication (2):
"Minimizing Logging to Facilitate Recovery of Tablespace", IBM Technical Disclosure Bulletin, vol. 29, No. 8, Jan. 1987.

CLAIMS:

1. In a computerized database system including a storage medium for storing data files, index files, and a recovery log having a plurality of files, and a CPU for I/O to said storage medium, a method executed by said system for recovering a database after an I/O error, comprising the steps of:

(a) writing log records during normal forward processing to said recovery log on said storage medium;

(b) traversing across said recovery log on said storage medium during RESTART routine processing after said I/O error for detecting committed and uncommitted transactions, said RESTART routine including a REDO sub-routine for redoing committed transactions recorded in said recovery log;

(c) detecting said written log records in said recovery log which are incomplete or failed;

(d) executing a recovery routine;

(e) executing an UNDO routine for undoing said detected uncommitted transactions;

(f) renaming said files on said storage medium having said I/O error;

(g) storing in a first file on said storage medium indicators of said renamed files that identify said renamed files as error files;

(h) accessing said first file for said indicators;

(i) preventing a subsequent RESTART routine from accessing said identified I/O error files in response to said accessing said first file;

(j) continuing said RESTART routine in response to said accessing said first file while preventing operations on said identified I/O error files during said continued RESTART routine;

(k) generating a request for access to one of said data files on said storage medium having a corresponding index file;

(l) accessing said one of said data files on said medium to determine whether said corresponding index file has been renamed thereby indicating said corresponding index file is an error index file;

(m) rebuilding said index file in response to said determination of said error index file.

First Hit Fwd Refs

Generate Collection Print

Search Forms
Search Results
Help
User Searches
Preferences
Logout

L9: Entry 20 of 24 File: USPT May 25, 1999

DOCUMENT-IDENTIFIER: US 5907848 A
TITLE: Method and system for defining transactions from a database log

Brief Summary Text (3):
The invention disclosed herein relates generally to database recovery systems. More particularly, the present invention relates to a method and system for defining transactions based on a database log, keeping track externally to the log of transactions in progress at any one time, and recovering from system failures by at least undoing any actions performed on the database as part of transactions which were in progress at the time of system failure.

Brief Summary Text (4):
Database recovery is an important function of a database system. Data in a database can be lost or damaged due to various types of failures, including physical disasters (fires, floods, etc.), computer system crashes, software or human error, and physical failures in the media upon which the database is stored. To prepare for such failures, a backup copy of the database is usually stored on a secondary media, and the backup copy is periodically updated to match the database. Also, a log or journal is maintained which keeps track of changes made to the database, and the log is used in the event of loss or damage to the database to reproduce the changes made to the database since an earlier backup.

Brief Summary Text (7):
During database recovery, the recovery program scrolls through the log looking for records having the START and COMIT/ABORT statements. For each completed transaction having START and COMMIT statements, the program retrieves the records in-between associated with the transaction and performs a redo or roll forward by updating the database with the actions in the transaction if such actions have not already been performed in the database. For incomplete transactions having a START statement with no COMMIT, or for transactions ending with an ABORT, the actions in the transaction are undone or rolled back if they had already been performed in the database. The use of checkpoints in the log helps identify which actions have been performed in the database.

Brief Summary Text (17):
There is thus a need for a method for defining transactions in the absence of commitment control and for a method for protecting transactions during database recovery using logs generated by programs lacking embedded logic for setting transaction boundaries. The present invention provides these and other advantageous functions.

Drawing Description Text (6):
FIG. 4 is a sample log record produced by a DB2/400 database management system;

Drawing Description Text (9):
FIG. 7 is a flow chart showing the process of recovering from database failure in accordance with one embodiment of the present invention.

Detailed Description Text (3):
With reference to the block diagram in FIG. 3, a database system 10 contains a

currently active database 12 which is operated upon by a number of users 14 and a backup database 16 which is stored in a secure location, contains an earlier version of the data in the database 12, and is periodically updated. The database 12 may be a relational database such as DB2 in which the data stored in the database 12 is perceived in the form of tables or files identified by table names or filenames and having groups or sets of data values each of which is identified by a name and contains data values of a certain data type. Each file contains a number of records or entries, each record containing a number of fields each having a data value from one of the groups. The structures of databases and the operation of database management systems are well known in the art and need not be described in detail herein.

Detailed Description Text (11):
The database system 10, transaction defining system 22, and particular elements thereof may be stored on a single mainframe or microcomputer system or on separate computer systems connected over a local network or remotely via a telecommunications link. The present invention has particular applicability to database systems operating on IBM's AS/400 environment, because many of such systems, including several versions of DB2, lack embedded commitment control and produce logs lacking transaction definitions. One skilled in the art will recognize that the present invention may be used on any database or other type of computer system, including any system that includes embedded commitment control.

Detailed Description Text (12):
A sample log record produced on an IBM AS/400 mainframe using facilities associated with a DB2/400 database is shown in FIG. 4. The record, which represents the type of record used in one embodiment of the invention, contains a number of fixed length fields which have been visually separated for convenience by the use of separate underlining in the Figure. The record also contains data values (non-underlined) which relate to data from the database record affected by the action. In addition, an offset ruler is provided above the log record to facilitate location of fields by their offset from the start of the record. The name of and type of data stored in each fixed length field can be obtained by reference to Field Description tables available from IBM Corp. However, a few specific fields will be discussed herein to provide a more complete understanding of the operation of the invention.

Detailed Description Text (23):
Once the log, template and job identifier data are specified, the log is opened and log records retrieved one at a time, step 62. Each log record is checked to determine whether it contains the specified job identifier data, step 64. If the record does not contain the specified job identifier data, the record is skipped and the next record retrieved. If the record relates to the job, the data values from the record which relate to data from the database are retrieved, step 66. In the sample DB2/400 log record shown in FIG. 4, this data is stored in the non-fixed length field at the end of the record.

Detailed Description Text (62):
Aside from the use and operation of the system and program for database recovery as described herein, the system may be used for database locking as well. As will be recognized by those of skill in the art, the index files which represent ongoing transactions may be used to identify records in the log, which in turn may be used to identify database files which are being used as part of transactions. The files can then be locked according to known locking schemes.

CLAIMS:

22. The method of claim 21 wherein the database comprises a plurality of tables, wherein each log record contains a name of a table in the database affected by a change to the database, and wherein the table names from records having one or more

matching data values are stored in the transaction template associated with the key value.

23. The method of claim 22 comprising displaying the table names stored in the transaction template and modifying the table names based on input received from the user.